Datorlaborationer, Programmering i C++ (EDA623)

Datorlaborationerna ger exempel på tillämpningar av det material som behandlas under kursen.

- *Uppgifterna i laborationerna ska lösas i par om två.* I samband med anmälan till till laborationerna får du möjlighet att ange vem du vill samarbeta med.
- *Laborationerna är obligatoriska.* Det betyder att du måste bli godkänd på alla uppgifterna under ordinarie laborationstid. Om du skulle vara sjuk vid något laborationstillfälle så måste du anmäla detta till kursansvarig lärare före laborationen (epost-adress finns på kursens hemsida).

  Om du varit sjuk bör du göra uppgiften på egen hand och redovisa den under påföljande laborationstillfälle. Det kommer också att anordnas en uppsamlingslaboration i slutet av kursen. Detta tillfälle erbjuds dock bara till dem som haft giltigt skäl för frånvaro på någon laboration eller som varit närvarande vid samtliga laborationer men, trots rimliga förberedelser, inte hunnit bli färdiga.

  Observera att detta är sista gången EDA623 ges. Nästa läsår kommer den att ersättas med en ny kurs, med delvis andra obligatoriska moment. De som inte blir klara med de obligatoriska laborationerna i höst (enligt ovan) kommer att hänvisas till att komplettera de moment som saknas då den nya kursen ges. Detta kan innebära att det blir delvis andra laborationsuppgifter som då skall redovisas.
- *Laborationerna kräver en hel del förberedelser.* I början av varje laboration finns anvisningar om förberedelser under rubrikerna Läsanvisningar och Förberedelser. Läsanvisningarna anger vilka avsnitt i läroboken som ska läsas. Under rubriken Förberedelser anges vilka av laborationsuppgifterna som *ska* lösas före laborationstillfället. Du ska också ha läst igenom de övriga uppgifterna och gärna försökt lösa dem. Det är inget krav att att du kommer med helt färdiga lösningar. Men det är ditt och din laborationspartners ansvar att ha förberett er så att ni bedömer att ni hinner bli klara under laborationen. Ni får naturligtvis under dessa förberedelser gärna kontakta kursasnvarig lärare om ni stöter på svårigheter.
- Du måste för varje laboration se till att laborationsledaren noterar dig som godkänd på listan på nästa sida.

Om du hittar någonting i uppgifterna eller andra anvisningar som är felaktigt eller oklart så meddela gärna detta till kursansvarig lärare.

Innehåll

Programmering i C++, godkända laborationsuppgifter

Skriv ditt namn och din namnteckning nedan:

Namn: .....

Namnteckning: .....

Godkänd laboration	Datum	Laborationsledarens namnteckning
1		
2		
3		
4		
5		
6		

Laboration 1

Mål: Du ska på de områden som behandlats under de första föreläsningarna: de grundläggande delarna av C++ samt tecken och texter.

Läsanvisningar

Läs igenom följande avsnitt i boken: 2.1–2.10, 3.1–3.4. Läs också på föreläsningsbilderna från föreläsning 1 och 2, som finns på kursens hemsida.

Förberedelser

CodeBlocks

På skolans datorer finns CodeBlocks installerat. Om du vill jobba med laborationen hemma och inte redan laddat ner CodeBlocks eller annan utvecklingsmiljö för C++ så finns anvisningar för detta på hemsidan.

Förberedelser för den första laborationen

Lös uppgifterna U1-U3 och U6-U8 nedan under rubriken ”Uppgifter”. Läs igenom uppgifterna U4-U5 och U9-U10. Uppgift U9 är frivillig.

Uppgifter

U1. Skriv om följande uttryck med if-satser:

```
x = (a > b) ? a : (b > c) ? b : c ;
```

Lägg till kod så att `x` innehåller maxvärdet av `a`, `b` och `c`. Testa det omskrivna uttrycket i följande kodavsnitt:

```
for (int a = 1; a <= 3; a++) {
  for (int b = 1; b <= 3; b++) {
     for (int c = 1; c <= 3; c++) {
         int x = 0;
         int m = a; if (b > m) m = b; if (c > m) m = c;

         // lägg in omskriven kod att testa här
         // x = (a > b) ? a : (b > c) ? b : c ;

         cout << "a b c x = " << a << " " << b << " " << c << " " << x;
         if (x != m) cout << " max = " << m;
         cout << endl;
     }
  }
}
```

U2. Skriv ett program som kontrollerar om och hur mycket man vunnit i ett lotteri av typ Joker. Programmet kan antingen själv generera det sexsiffriga vinstnumret (slumptal) eller också låta användaren mata in det först. Man ska sedan kunna mata in flera lottnummer för att kontrollera eventuell vinst. Vinsten är 10 kr om första (entals-) siffran överensstämmer, 100 kr om de två första överensstämmer osv. Det finns även en extra utdelning på 250 kr om de tre första siffrorna förekommer i samma ordning någon annanstans i vinstnumret (står de först vinner man 1000 kr).

U3. I vissa olympiska gymnastikgrenar beräknas den tävlandes resultat som medeltalet av erhållna domarpoäng efter att man har tagit bort den högsta och lägsta poängen. Skriv ett program som läser in domarpoängerna (3 eller fler) och beräknar resultatet enligt ovan.

U4. Ändra uppgiften ovan så att domarpoängen läses in från en fil och gör en lämplig fil. OBS! Decimalpunkt.

U5. Skriv ett program som slumpar fram en av tre möjliga utfall och skriv ut texten STEN, SAX eller PÅSE beroende på utfallet. Lägg sedan till att man får mata in ett av alternativen först och att programmet talar om vem som vann.
(STEN vinner över SAX, SAX vinner över PÅSE och PÅSE vinner över STEN)

U6. Skriv ett program som läser in en sträng och sen skriver ut den baklänges.

U7. Skriv ett program som avgör om ett angivet bilnummer är giltigt. Utgå från den enkla formen med tre bokstäver följt av tre siffror (t.ex. HEJ345).

U8. Det s.k. Caesarchiffret går ut på att rotera alfabetet ett visst antal positioner och på så sätt obegripliggöra ett meddelande. Skriv ett program som läser in ett meddelande i form av en sträng på högst 80 tecken. Meddelandet krypteras genom att förskjuta varje bokstav 13 positioner i alfabetet (engelskt alfabet med 26 bokstäver). Efter ’Z’ börjar alfabetet om igen från ’A’. Alla gemener (små bokstäver") i meddelandet görs först om till versaler (stora bokstäver") innan själva krypteringen genomförs.

U9. Modifiera föregående uppgift så att krypteringen görs genom en godtycklig permutation av bokstäverna A-Z. Enklast är att lagra det permuterade alfabetet i en sträng, t.ex. "QWERTYUIOPASDFGHJKLZXCVBNM".

U10. (bokens övning 3.9): De romerska siffrorna anges med bokstäverna I, V ,X ,L ,C ,D och M som står för 1, 5, 10, 50, 100, 500 respektive 1000. Deklarera först en string-variabel som innehåller de romerska siffrorna och sedan en tabell (ett fält) som kan användas för att översätta en romersk siffra till ett vanligt heltal (t.ex. L till 50). Skriv sedan ett program som läser in ett romerskt tal till en string-variabel och som översätter det romerska talet till ett vanligt heltal. Om användaren t.ex. skriver MCMXLIX skall programmet skriva ut 1949. Programmet skall ge en felutskrift om det inmatade romerska talet är felaktigt. I ett romerskt tal gäller att om en romersk siffra P står omedelbart till vänster om en annan romersk siffra Q och om P betecknar ett mindre tal än Q, så skall värdet av P subtraheras från det totala talet (LIX betyder t.ex. 59), annars skall P adderas till det totala talet (LXI betyder 61).

Laboration 2

Mål: Du ska träna på att konstruera och anropa egna funktioner i C++. Du ska också träna på användning av pekare i C++.

Läsanvisningar

Läs kap 4–5 i läroboken. Läs också föreläsningsbilder från de föreläsningar, som behandlar motsvarande avsnitt. Dessa finns på kursens hemsida.

Förberedelser

Läs igenom den inledande texten under rubriken ”Uppgifter” nedan och lös uppgift U1-U3, U5, U8, U10 och U12-U13. Läs igenom och sätt dig in i uppgifterna U4, U6-U7, U9, U11 och U14. Uppgifterna U3-U4 och U10 är frivilliga.

Uppgifter

I dessa övningar ingår att skriva en `main`-funktion för test av funktionerna.

U1. Skriv en funktion `delbar` som har 2 heltalsparametrar (`a` och `b`) och returnerar `true` om `a` är jämnt delbar med `b`, annars `false`.

U2. Skriv en funktion `kvadratkubtabell` som har en värdeparameter `n` av typen `int` och som skriver ut en tabell med heltalen från `1` till `n` samt deras kvadrater och kuber. Förse tabellen med lämplig rubrik.

U3. Det finns en standardfunktion, `rand` (deklareras i `<cstdlib>`), som returnerar s k pseudoslumptal, heltal i intervallet 0 till `RAND_MAX` (systemberoende).
Skriv en funktion, `randInt(min, max)`, som i stället returnerar ett heltal i intervallet `min..max` (låt funktionen utnyttja `rand` för att åstadkomma detta).

U4. Skriv en funktion `downcase` som tar in en C-sträng och ändrar alla versaler (STORA BOKSTÄVER) i strängen till gemener (små bokstäver).

U5. Ett polynom av fjärde graden kan skrivas
$$p(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$
En funktion `poly` ska skrivas som beräknar polynomfunktionens värde för godtyckligt val av `x`. Polynomkoefficienterna läggs i en vektor som får utgöra en av inparametrarna till funktionen.

U6. Skriv en funktion som tar en heltalsvektor som inparameter samt ökar varje heltal i vektorn med `1`. Använd inparametern som utparameter också samt utnyttja referensanrop.

U7. Skriv två olika varianter av en funktion som beräknar summan av kvadraterna på alla heltal från `1` till ett givet tal `n`. Den ena varianten skall vara iterativ och den andra rekursiv.

U8. Gör en egen version `mystrcat` av funktionen `strcat(*char s,const *char t)` som lägger till strängen `t` till slutet av strängen `s` (konkatenering).

U9. Skriv en funktion, `char *strreverse(char *s)` , som vänder på innehållet i C-strängen `s`. Om man t ex anropar funktionen med ’’123’’ funktionen ge strängen ’’321’’ som resultat.

U10. I C-biblioteket `<cstring>` finns funktionen `int strcmp(const char* s1, const char* s2);` Denna funktion jämför C-strängarna `s1` och `s2`. Resultatet är `< 0` om `s1` är mindre än `s2`, `0` om strängarna är lika och `> 0` om `s1` är större än `s2`. Jämförelsen sker tecken för tecken i ASCII-ordning. Exempelvis är `''Berit''` mindre än `''Bertil''` eftersom det fjärde tecknet skiljer sig och `'i'` är mindre än `'t'`. `''Bert''` är mindre än `''Bertil''`, eftersom första strängen är kortare än den andra (och de gemensamma tecknen är lika). Implementera `mystrcmp` med samma beteende som `strcomp`, utan att utnyttja några standardfunktioner.

U11. Skriv funktionen `void byt(int *a, int *b)` som byter plats på `a` och `b` om `a > b`. Använd sedan funktionen för att sortera en heltalsvektor `v` med `n` element enligt följande princip: Jämför `v[k]` med varje element med index `>k` och byt om `v[k]` är störst. Låt `k` gå från `0` till `n-1`.

U12. Skriv ett program som frågar efter ett meddelande (textsträng) och sen kodar om den till morsesignaler genom att skriva ut morsesymbolerna (med blanktecken emellan). Morsealfabetet ges av följande tabell:

```
A .-     B -...   C -.-.   D -..    E .      F ..-.
G --.    H ....   I ..     J .---   K -.-    L .-..
M --     N -.     O ---    P .--.   Q --.-   R .-.
S ...    T -      U ..-    V ...-   W .--    X -..-
Y -.--   Z --..   Å .--.-  Ä .-.-   Ö ---.
```

U13. Skriv ett program som slumpar ut talen 0-15 i en 4x4-matris och därefter skriver ut den. Exempelutskrift:

```
 3  7  9  8
11  6  1  4
 5  2 14 10
12  0 15 13
```

U14. Komplettera föregående uppgift så att man kan spela det s.k. 15-spelet. Talen 1-15 tolkas här som brickor som kan flyttas om det är en lucka intill (0). Eventuellt blir pariteten fel (t.ex. alla rätt utom två brickor som har bytt plats) och då saknar spelet lösning. Detta problem bortser vi emellertid ifrån vid slumpningen (i förra uppgiften) för enkelhets skull. Dialogexempel (med utgångsposition enligt ovan):

```
Vilken bricka ska flyttas? 12
  3  7  9  8
 11  6  1  4
  5  2 14 10
  0 12 15 13
```

Laboration 3

Mål: Du ska på att konstruera och använda egna klasser i C++. Du ska också träna på användning av datamedlemmar, medlemsfunktioner, konstruktorer, destruktorer, uppräkningstyper och överlagrade operatorer för för dessa.

Läsanvisningar

Läs kap 7–8 i läroboken. Läs också föreläsningsbilder från de föreläsningar, som behandlar motsvarande avsnitt. Dessa finns på kursens hemsida.

Förberedelser

Läs igenom den inledande texten under rubriken ”Uppgifter” nedan och lös uppgift U1, U3a och U4. Läs igenom och sätt dig in i uppgifterna U2, U3b och U5. Uppgift U2b-c är frivillig.

Uppgifter

U1. a) Konstruera en `Rektangel`-klass som har rektangelns längd och bredd (båda flyttal) som datamedlemmar. Klassen ska ha följande medlemsfunktioner:

- `sattVarden` som ger datamedlemmarna värden (via parametrar)
- `area` som räknar ut och returnerar rektangelns area (som funktionsresultat)
- `visa` som skriver ut rektangelns längd och bredd

Inför en `main`-funktion som skapar ett `Rektangel`-objekt och använder medlemsfunktionerna. Rektangelns längd och bredd läses in från tangentbordet.

b) Komplettera `Rektangel` med medlemsfunktionen `omkrets` som returnerarrektangelns omkrets. Testa funktionen i `main`.

c) Inför ytterligare en medlemsfunktion, `forstora`, som har ett heltal (`faktor`) som parameter och som multiplicerar rektangelns längd och bredd med detta heltal (medför att rektangeln förstoras). Komplettera `main` så att den använder funktionen.

U2. Skriv en klass, `Mynt`, som kan användas för att simulera myntkast. Klassen skall ha de publika (synliga) medlemsfunktionerna:

```
Mynt()             // Initierar myntobjekt så att en slumpmässig sida
                   // kommer upp (krona/klave)
void kasta()       // Simulerar ett myntkast
void visa()        // Skriver texten "Krona" eller "Klave", beroende på
                   // myntets tillstånd, dvs vilken sida som är vänd uppåt
Myntsida uppsida() // Returnerar myntets tillstånd
```

- Representera de två tillstånden med en uppräkningstyp,
  `enum Myntsida krona, klave`
  som definieras i klassens publika (synliga) del
- Klassens datamedlemmar skall vara privata
- Lägg klassdefinitionen i en fil med namnet `mynt.h`, och definitionerna av medlemsfunktionerna i `mynt.cpp`

a) Skriv ett program, som låter användaren göra upprepade kast med ett mynt. Resultatet av varje kast skall skrivas ut.

b) Skriv ett program som simulerar n kast (där n är ett positivt heltal) med två mynt (kasten visas) och rapporterar hur många av utfallen som blev lika.

c) Generalisera föregående uppgift så att m mynt kan användas (använd en dynamiskt allokerad array av myntobjekt).

U3. I en datoriserad svensk-engelsk ordbok beskrivs ordpar av följande klass:

```
class Word {
public:
   Word(char *sw, char *eng);   //Svenskt sw, engelskt eng
   ~Word();
   const char *get_sw() const;  //Hämta svenskt ord
   const char *get_eng() const; //Hämta engelsk översättning
private:
   char *swedish;
   char *english;
};
```

a) Implementera klassen `Word`.

b) Skriv en klass `Dictionary` som beskriver ordboken. Operationer: Lägg in ett nytt svenskt ord (med engelsk översättning), tag reda på den engelska översättningen av ett svenskt ord. Du får använda valfri metod för att lagra ordobjekten i ordboken. Följande funktioner ur C-biblioteket `<cstring>` kan komma till användning:

```
int strlen(char* s);
void strcpy(char* dest, char* source);
int strcmp(char* s1, char* s2);
```

U4. a) Konstruera en klass Kvadrat i vilken ingår:

- Datamedlem: kvadratens sida (heltal)
- Konstruktorer: Standardkonstruktor, en konstruktor med kvadratens sida som argument, kopieringskonstruktor (om sådan behövs, om inte motivera!)
- Medlemsfunktion som returnerar kvadratens area

b) Överlagra följande operatorer:

- Tilldelningsoperatorn (om det behövs)
- Öknings- och minskningsoperatorerna ++ och - (prefixvarianten räcker).
- Låt ++ betyda ökning av kvadratens sida med `1` och - minskning med `1` (om inte sidan blir `< 0`)
- + och - som betyder ökning resp minskning av kvadratens sidlängd med ett heltal (sidan får dock ej bli `< 0`)
- Jämförelseoperatorerna (`<`, `>`, `==` osv) med vänfunktioner

Skriv även ett testprogram.

U5. Implementera klassen `Personnummer`. Den ska använda teckenarray eller charpekare för att lagra personnumret och ha en boolsk variabel som anger om personnumret är OK eller inte. Överlagra operatorerna « och » för in- och utmatning av personnummer. kontrollera personnumret med en privat funktion (hur långt kontrollen ska drivas bestämmer du själv, men den ska åtminstone avgöra om kontrollsiffran är riktig). Överlagra även operatorn `!` så att den returnerar `true` om personnumret är felaktigt (utnyttja den boolska variabeln i Personnummer). Klassen ska kunna användas på följande sätt:

```
Personnummer persnum;
cout << "Ange personnummer: ";
cin >> persnum;
while (!persnum) {
  cerr << "Personnumret är felaktigt!\n";
  cout << "Ange personnummer: ";
  cin >> persnum;
}
cout << "Personnumret " << persnum << " är korrekt.\n";
```

Du bestämmer själv om personnummer bara ska innehålla siffror eller om det även ska (eller får) innehålla ett bindestreck före de 4 sista siffrorna.
Kontrollsiffran i ett personnummer kan bestämmas på följande sätt:

1. Utgå från de första 9 siffrorna och multiplicera siffrorna på udda plats (1, 3 osv) med 2 och siffrorna på jämn plats (2, 4 osv) med 1.
2. lägg ihop alla *siffrorna* i dessa produkter (dvs 12 räknas som 1+2=3).
3. Tag entalssiffran i den framräknade summan. Kontrollsiffran är då 10-entalssiffran, utom om entalssiffran är 0, då också kontrollsiffran är 0.

Laboration 4

Mål: Du ska träna på arv, virtuella funktioner och aggregat. Du ska också generera och fånga exceptionella händelser samt filhantering i C++.

Läsanvisningar

Läs kap 9–11 i läroboken. Läs också föreläsningsbilder från de föreläsningar, som behandlar motsvarande avsnitt. Dessa finns på kursens hemsida.

Förberedelser

Läs igenom den inledande texten under rubriken ”Uppgifter” nedan och lös uppgift U1-U2 och U5-U6. Läs igenom och sätt dig in i uppgifterna U3-U4 och U6-U7. Uppgifterna U1 och U7 är frivilliga.

Uppgifter

I denna uppgift ska du

U1. En cylinder kan uppfattas som en cirkel med höjd, dvs man kan låta den ärva från en cirkel. Skapa en sådan klass, Cylinder, utgående från klassen Circle nedan.

```
class Circle {
public:
  Circle(double r=0) : radius(r) {}
  double getArea() const;
  double getRadius() const { return radius; }
private:
  double radius;
};
// Implementering av medlemsfunktioner:
double Circle::getArea() const {
  return 3.1416*radius*radius;
}
```

Förutom en konstruktor, som har cirkelns radie och cylinderns höjd som parametrar, ska klassen ha följande medlemsfunktioner: `getVolume()` som returnerar cylinderns volym samt `getHeight()` som returnerar cylinderns höjd. Skriv en `main`-funktion som testar `Cylinder`-klassen.

U2. Skapa en abstrakt basklass, `Pet`, som förutom standardkonstruktor och virtuell destruktor, har den rena virtuella medlemsfunktionen `speak()`. De tre subklasserna `Dog`, `Cat` och `Bird` har egna `speak`-versioner som ger ifrån sig lämpliga djurläten.
Skriv en `main`-funktion som först skapar ett objekt av vardera `Dog`, `Cat` och `Bird`, varefter dessa utnyttjas för att initiera ett fält av pekare till basklassen `Pet`. Inför även en pekarfält, där ingående djur (ett av vardera slaget) är dynamiskt allokerade. Använd upprepningssats för att låta djuren i fälten tala (hitta på lämpliga läten). Vad händer om man försöker skapa ett objekt av typen `Pet`?

U3. Denna uppgift går ut på att skapa en bostad (bestående av ett antal rum) genom att kombinera olika rumstyper. Utgå från basklassen `Rum` (se nedan) och skapa subklasserna `Kok`, `Badrum`, `Vardagsrum` och `Sovrum`, med datamedlemmar enligt:

```
Kok              harDiskmaskin och harFrys
Badrum           harDusch
Vardagsrum       harBalkong
Sovrum           antalGarderober
```

Konstruera sedan klassen `Bostad` (alla bostäder består av ett kök, ett badrum, ett vardagsrum och ett variabelt antal sovrum).
Klassen Rum:

```
class Rum {
private:
   int yta;
public:
   Rum(int y) : yta(y) {}
   virtual ~Rum() {}
   virtual void skriv() const { cout << "Yta: " << yta; }   };
```

Skriv klasserna `Bostad`, `Kok`, `Badrum`, `Sovrum` och `Vardagsrum`. Förutom konstruktorer och destruktorer ska samtliga klasser ha medlemsfunktionen `skriv()` som ger rumsinformation eller info om en hel `Bostad` (ett antal rum).

Följande testprogram skall vara körbart:

```
int main() {
  int sovrumsstorlek[] = {16, 14}; // 2 sovrum
  int garderober[] = {4, 3};
  Bostad minLya(12, false, true, // kök
                4, true, // badrum
                24, false, // vardagsrum
                sovrumsstorlek,
                garderober,
                sizeof(sovrumsstorlek)/sizeof(int));
  minLya.skriv();
}
```

och då ge utskriften:

```
Kök: Yta: 12, Diskmaskin: Nej, Frys: Ja
Badrum: Yta: 4, Dusch: Ja
Vardagsrum: Yta: 24, Balkong: Nej
Sovrum: Yta: 16, 4 garderober
Sovrum: Yta: 14, 3 garderober
```

U4. a) Välj ut någon av medlemsfunktionerna eller operatorerna i klassen `Vektor` (från föreläsningarna) där `assert` används och skriv om felhanteringen så att undantag genereras istället. Välj själv lämplig undantagstyp (ev. någon ny).

b) Fånga genererade undantag i ett huvudprogram

U5. Skriv ett program som avgör om innehållet i två filer är lika. Dialog:

```
Ange första filens namn: prog1.cpp
Ange andra filens namn: prog2.cpp
Filerna är inte lika
```

Begreppet likhet skiljer sig mellan textfiler (lika många rader med alla raderna parvis lika) och binära filer (lika många bytes med alla bytes parvis lika). Implementera den binära varianten.

U6. Vi har en textfil med namn på studerande i en kurs vid Grönköpings högskola (Internetdomän gronkoping.se). Varje rad i filen innehåller för- och efternamn för en person. Förnamnet skiljs från efternamnet med ett blanktecken, inga andra blanktecken finns i filen. Nu vill vi utifrån denna fil producera en ny fil med studenternas epostadresser.

Vi ska alltså läsa en person i taget från den gamla filen (som heter `namnlista.txt`) och utifrån personens namn generera en epostadress som sparas i en ny fil (med namnet `email.txt`). Exemplet nedan visar hur epostadresserna ska se ut. Om `namnlista.txt` innehåller

```
Arne Adamsson
Lena Evander
Orvar Persson   Petra Olsson
```

ska `email.txt` bli

```
Arne.Adamsson@gronkoping.se
Lena.Evander@gronkoping.se
Orvar.Persson@gronkoping.se
Petra.Olsson@gronkoping.se
```

OBS: Omdirigering av `cin` och `cout` är ej tillåten.

U7. Olika typer av filer har olika fördelning av tecken. Detta kan användas för att ta reda på av vilken typ en viss fil är, speciellt om filnamnet är vilseledande. Visserligen kan man då identifiera speciella teckenkombinationer som ska finnas i speciella positioner i filen (signaturer vilka lagras i en tabell). Ett alternativ till detta är att göra en enkel karakterisering av teckenfördelningen (fileprint analysis). En noggrannare analys används för att komplettera traditionell virus- och maskdetektering. I denna uppgift delar vi (lite grovt) in intervallet 0-255 (ASCII-kod) i 6 olika kategorier: 0, 1-31 (kontrolltecken förutom 0), 32 (blanktecken), 33-127 ("vanligatecken) , 128-254 respektive 255. Skriv ett program som läser igenom en fil och sedan skriver ut procentuell fördelning av de 6 olika kategorierna. *Exempeldialog*:

```
Filnamn: d:\winnt\notepad.exe
   Kod      %
---------------------
      0:  33.91    1-31:  12.43
     32:   0.93
 33-127:  27.40
128-254:  20.22
    255:   5.10
```

Laboration 5

Mål: Du ska på att använda containerklasser i C++. Du ska också implementera egna containerklasser med hjälp av pekare och fält.

Läsanvisningar

Läs kap 12–13 i läroboken. Läs också föreläsningsbilder från de föreläsningar, som behandlar motsvarande avsnitt. Dessa finns på kursens hemsida.

Förberedelser

Läs igenom den inledande texten under rubriken ”Uppgifter” nedan och lös uppgift U1. Läs igenom och sätt dig in i uppgifterna U2-U5. Uppgift U4 är frivillig.

Uppgifter

I denna uppgift ska du

U1. Denna uppgift går ut på att simulera en bankomatkö med användning av en container av slaget queue. Kön ska innehålla kunder av följande slag:

```
class Kund {
  int atid; // ankomsttid
  int btid; // betjäningstid
public:
  // medlemsfunktioner
}
```

Förutom konstruktorer (som initierar datamedlemmarna på lämpligt sätt) behöver `Kund`-klassen medlemsfunktioner för att avläsa ankomsttid och betjäningstid. Betjäningstiden slumpas fram (1-4 minuter med lika stor sannolikhet). Det är lämpligt (men ej nödvändigt) att utföra detta i en konstruktor. Själva simuleringen går till på så sätt att användaren får ange hur många timmar som ska simuleras och det genomsnittliga antalet kunder/timme (högst 60). Sedan används en loop där varje varv får motsvara 1 minut, varvid man kan göra på följande sätt:

1. Slumpa fram nya kunder (så att det genomsnittliga antalet kunder/timme uppfylls).
2. Om en ny kund anlänt placeras denne i kön.
3. Om bankomaten är ledig börjar man betjäna kunden som står främst i kön (om det finns någon kund att betjäna).
4. Minska återstående betjäningstid med 1 för den kund som betjänas.

Programmet ska hålla reda på totala antalet betjänade kunder och total kötid. För att det ska vara möjligt att beräkna den genomsnittliga kölängden när själva simuleringen är klar behöver man dessutom varje minut avläsa och summera den aktuella kölängden (summan sparas en särskild variabel för ackumulerad kölängd). När simuleringen är klar redovisas statistik.

U2. Skriv ett program som framslumpar positiva heltal (0-9999) och insätter dessa först i en enkellänkad lista. Skriv ut listan och bestäm största talet (låt funktioner utföra jobbet).

U3. Skriv en funktion som söker efter ett tal i en dubbellänkad lista av det slag som beskrivs i C++ direkt. Funktionen ska ha listan och det sökta talet som parametrar och returnera en pekare till noden med talet om det hittades och en NULL-pekare annars.

U4. Läroboken implementerar en stack med en länkad lista, men det går även utmärkt att utnyttja ett fält till detta. Implementera en sådan stack med utgångspunkt från nedanstående definition. Lägg märke till att stackens storlek ska kunna bestämmas när stacken skapas och att pop fungerar på annat sätt än bokens version.

```
#include <stdexcept>
class Stack {
private:
  int maxSize; // Stackens maximala storlek
  int *data;   // Data
  int num;     // Antalet element stacken
public:
  Stack(int max=100);
  ~Stack();
  void clear();
  bool empty() const;
  bool full() const ;
  int pop() throw (length_error); // Hämta+avlägsna element
  void push(int value) throw (length_error);
};
```

U5. Det är även möjligt att implementera köer med fält, fast det är knivigare. Problemet är att i en kö sker operationerna i båda ändarna. Ett sätt att lösa detta är att använda en slags rullande buffert (cirkulär kö), där man börjar om från andra änden när man nått buffertens slut. Implementera en sådan kö med utgångspunkt från följande definitioner:

```
#include <stdexcept>
class Queue {
private:
  int maxSize; // Köns maximala storlek
  int *data;   // Data     int head, tail;
public:
  Queue(int max=100);
  ~Queue();
  void clear();
  bool empty() const;
  bool full() const;
  int dequeue() throw (length_error); //Hämta+avlägsna elem
  void enqueue(int value) throw (length_error);
  int length();
};
```

Tips: Det kan vara lite trixigt att få ihop det hela, särskilt gäller detta kontrollen om kön är full. Enklast hanteras detta med en räknare som håller reda på antalet element i kön (det är tillåtet att komplettera specifikationen med en sådan variabel).

Laboration 6

Mål: Du ska på att konstruera och använda egna mallar (templates) i C++. Du ska också träna på användning av `struct`, `union`, bit-operatorer och bit-fält.

Läsanvisningar

Läs kap 14–15 i läroboken. Läs också föreläsningsbilder från de föreläsningar, som behandlar motsvarande avsnitt. Dessa finns på kursens hemsida.

Förberedelser

Läs igenom den inledande texten under rubriken ”Uppgifter” nedan och lös uppgift U2, U4, U5 och U7. Läs igenom och sätt dig in i uppgifterna U1, U3, U6 och U8. Uppgifterna U3-U4 är frivilliga.

Uppgifter

I denna uppgift ska du

U1. Skriv en funktionsmall, `skrivUtFalt()`, som har ett fält av godtycklig typ som parameter (fler parametrar kan krävas) och skriver ut fält på följande form:

```
[ 9, 7, 5, 3, 1 ]    // Fält av int
[ Kalle, Eva, Nisse] // Fält av string-objekt
```

Visa hur mallen används genom att skriva ut innehållet i fält av det slag som visas i exemplen ovan.
Vad krävs för att ett fält ska kunna skrivas ut med funktionsmallen?

U2. I lärobokens (C++ direkt) kapitel 5 (övning 18) beskrivs en effektiv sorteringsalgoritm som kallas *quicksort*. Utgå från algoritmen (eller lösningen till övningen) och skriv en funktionsmall (`quicksort.h`) som kan sortera godtyckliga datatyper med denna metod. Skriv också ett testprogram. Vilka egenskaper måste data som ska sorteras med funktionsmallen ha? Hur kan man komma förbi dessa?

U3. I C++ direkt sid 162 (3:e upplagan) finns en funktion som utför binärsökning i ett `int`-fält. Gör om funktionen till en funktionsmall och testa mallen med t ex ett `double`- och ett `int`-fält. Tänk dock på att binärsökning kräver att data är sorterade.

U4. I tidigare uppgift implementerades en stack med hjälp av ett fält. Gör om den till en klassmall. Skriv även ett testprogram som inför stackar med heltal och flyttal (värdena kan t ex slumpas fram).

U5. En tidigare uppgift gick ut på att implementera en kö med hjälp av ett fält. Gör om denna klass till en klassmall. Testa kön med framslumpade heltal.

U6. Skriv ett program som utnyttjar klassmallen i förra uppgiften för att hantera en kundkö där en kund representeras av en `struct`:

```
struct Kund {
  char namn[35];
  double belopp; // betalning
}
```

Kundernas (namn och belopp) läses in och placeras i kön. När en kund betjänas tas han eller hon bort från kön och får betala beloppet. Programmet ska hålla reda på hur många kunder som betjänats och totala försäljningssumman. Låt det hela ska styras via en meny där man får välja mellan att köa en ny kund, betjäna en kund eller avsluta. Antalet betjänade kunder och totalsumman redovisas löpande efter varje betjänad kund.

U7. Det finns många algoritmer för att komprimera data, dvs lagra data på ett mindre minneskrävande sätt. De flesta metoderna utnyttjar olika varianter av bitkodning.
Antag att vi vill behöver lagra datum. Ett naturligt sätt att representera datum är att utnyttja heltal (`int`): år, månad och dag, vilket kräver $3 * 4 = 12$ bytes om man utnyttjar vanliga (32 bitars) heltal för lagringen. Vi vill nu lagra detta på ett så minnessnålt som möjligt.
Vi kan enkelt halvera minnesåtgången genom att utnyttja 16 bitars heltal i stället för vanliga 32 bitars heltal. Fast minnesbehovet kan ytterligare reduceras om vi inser att dag och månad aldrig kan bli större än 31 resp 12, dvs 1 byte räcker för vardera (totalt $2+1+1 = 4$ bytes).
I själva verket kan vi med bitkodning få rum med alltihop i 2 bytes! För att inse detta gör vi följande observationer:

- För tal i intervallet 0..99 (årtalet) räcker 7 bitar (med 7 bitar klaras 0..127)
- För 1..12 (månadsnumret) räcker 4 bitar (klarar 0..15)
- För 1..31 (dagnumret) räcker 5 bitar (klarar 0..31)

Summa summarum: $(7+4+5) = 16$ bitar, vilket är två bytes, dvs $1/6$ av det ursprungliga minnesbehovet, en avsevärd besparing. Vi kan packa de tre datumkomponenterna i ett 2-bytes heltal enligt följande (Å=år, M=månad, D=dag):

Högre byte	Å	Å	Å	Å	Å	Å	Å	M
Lägre byte	M	M	M	D	D	D	D	D

a) Skriv en funktion, `packedDate` som returnerar ett packat datum enligt ovan. Dagnumret, månadsnumret och årtalet ska vara värdeparametrar.

b) Skriv en funktion, `unpackDate`, med ett packat datum som värdeparameter och med dag, månad och årtal i form av heltal som utparametrar.
Funktionen ska packa upp och leverera datum uppdelat i dess komponenter.

U8. Internt i datorn kan de i ett heltal ingående bytena lagras på olika sätt. Vanligast är rättvänd och bakvänd ordning (dvs med den mest signifikanta byten antingen först eller sist), men det finns även andra varianter,t ex kan bytena grupperas 2 och 2 och dessa 2- grupper lagras i rättvänd eller bakvänd ordning.

Rättvänd	4 bytes	12345678	2 bytes	1234	5678	1 bytes	12	34	56	78
Bakvänd	4 bytes	12345678	2 bytes	5678	1234	1 bytes	78	56	34	12

Att veta hur det ligger till kan vara viktigt, t ex när binära datafiler ska överföras från en dator till en annan. Ett sätt att undersöka hur det förhåller sig är att studera hur de olika delarna av ett värde lagras i minnet. Skriv ett program som i hexadecimal form skriver ut ett (4 bytes) heltal på tre olika sätt (som ett vanligt heltal, som två 2-bytes heltal och som fyra 1-bytes heltal). Utnyttja en `union` för ändamålet.